



File System Introduction

Title and content

- What is File System
- Method
- Aspects/Objects of FS
- Data Structure
- Multiple FS Support
- Virtual File System
- POSIX
- Demo

In [computing](#), **file system** or **filesystem** (often abbreviated to **fs**) is a **method** and **data structure** that the operating system controls how data is [stored](#) and retrieved.

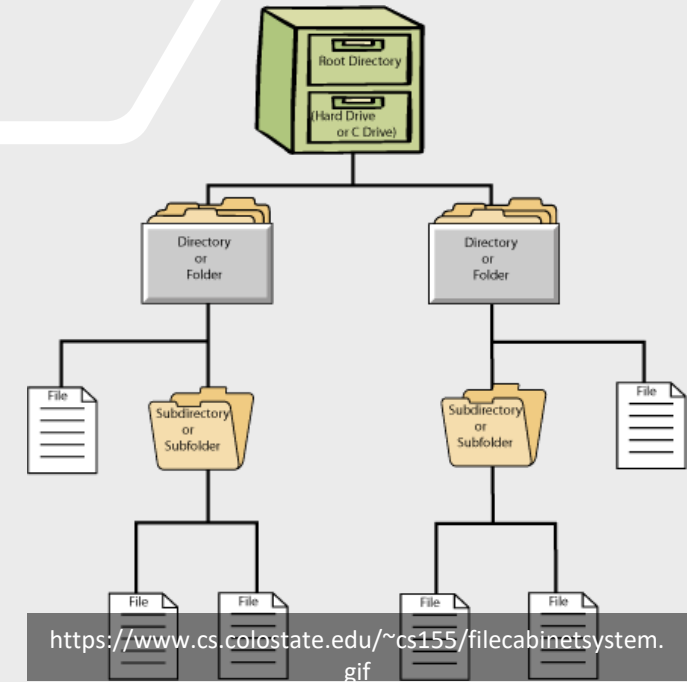
Without a file system, data placed in a storage medium would be **one large body of data** with no way to tell where one piece of data stops and the next begins.

From https://en.wikipedia.org/wiki/File_system

What is File System

What is File System

- I can make a file with specified name.
- I can make a directory with specified name.
- The file system is a tree structure containing files.
- ...
- I can save and modify data in a file.
- I can make a link to a file.
- A file has an owner and access permissions.
- ...
- Files (data) is persistent on storage device.
- ...
- I can code to open/read a file.

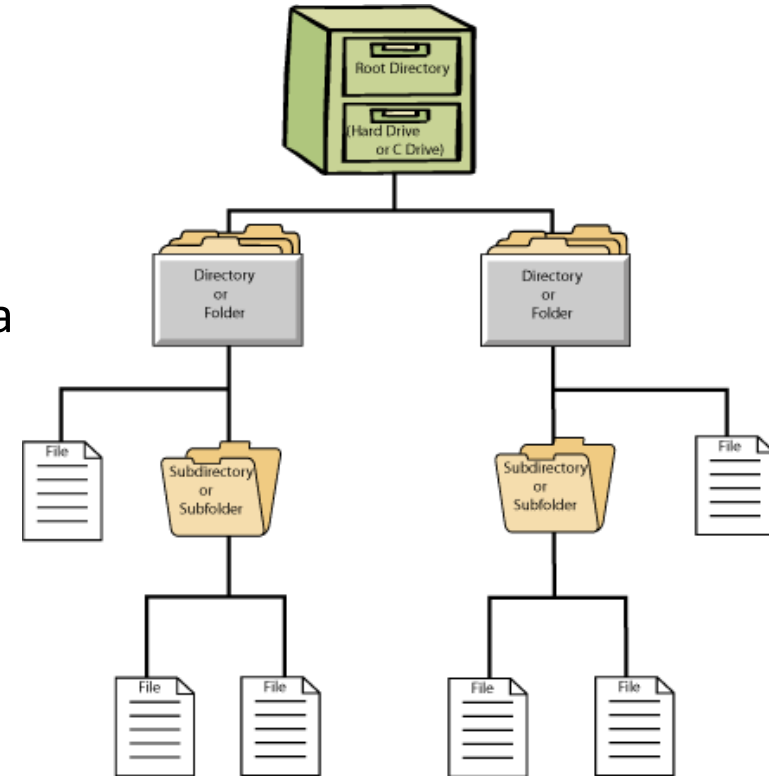


Method

How to manage Data (Large Data) ?

- Separating the data into pieces
- Giving each piece a name, each group of data is called a "file"
- Building mapping between name and data
- Access/Control via system calls (syscalls)

From https://en.wikipedia.org/wiki/File_system



Aspects/Objects of FS

Space management

- How to layout and utilize the storage device
- Block Size: unit size in FS
- Sector Size: unit size in storage device

Filenames

- used to identify a storage location in the file system
- encoding

Directories

- tables of files inside it

Metadata

- bookkeeping information (such as info associated with each file)

Data Structure

Filesystem

- A superblock: a record of the characteristics of a filesystem (*size, block size, inode tables, disk block map, usage information, size of block groups...*)

Space management

- Blocks Free or not: Bitmaps

Filenames

- Char[] & Encoding

Directories

- Just a file with “type=directory”

Metadata

- File/Directory: Index node (inode), storing attributes and disk block locations
- Blocks: free space bitmap
- ...

Inode – Reason for ‘I’ in “inode”

In 2002, the question was brought to Unix pioneer [Dennis Ritchie](#), who replied:^[4]

In truth, I don't know either. It was just a term that we started to use. "Index" is my best guess, because of the slightly unusual file system structure that stored the access information of files as a flat array on the disk, with all the hierarchical directory information living aside from this. Thus the i-number is an index in this array, the i-node is the selected element of the array. (The "i-" notation was used in the 1st edition manual; its hyphen was gradually dropped.)

Data Structure

- In-Memory
 - Runtime
- On-Disk
 - Persistent
























inode.h



dinode.h

Data Structure: UFS - Unix file system

 README.acfs	 dirhash.h	 ufs_dirhash.c	 ufs_lookup.c
 README.extattr	 extattr.h	 ufs_extattr.c	 ufs_quota.c
 acl.h	 inode.h	 ufs_extern.h	 ufs_vfsops.c
 dinode.h	 quota.h	 ufs_gjournal.c	 ufs_vnops.c
 dir.h	 ufs_acl.c	 ufs_inode.c	 ufsmount.h
	 ufs_bmap.c		

From <https://github.com/freebsd/freebsd-src/tree/main/sys/ufs/ufs>

Multiple FS Support

Different Filesystem has different characteristics or features, data structures.

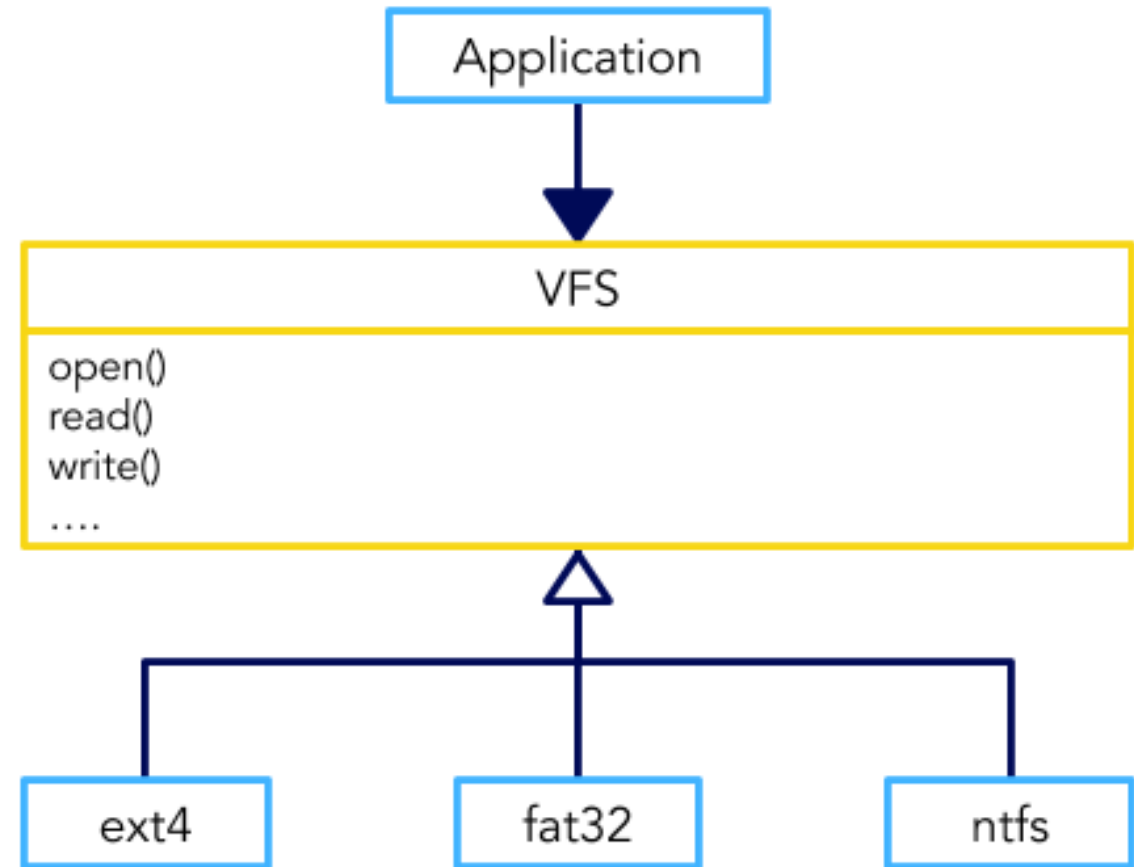
```
vagrant@vagrant-ubuntu-trusty-64:~$ mount -l  
/dev/sda1 on / type ext4 (rw) [cloudimg-rootfs]  
proc on /proc type proc (rw,noexec,nosuid,nodev)  
...  
vagrant on /vagrant type vboxsf (uid=1000,gid=1000,rw)
```

open(*filename*, *flags*) – each FS can have its specified functions ?

Virtual File System - VFS

The Virtual File System (also known as the Virtual Filesystem Switch) is the software layer in the kernel that provides the filesystem interface to userspace programs.

It also provides an abstraction within the kernel which allows different filesystem implementations to coexist.



Virtual File System - VFS

4 Main Abstracted Objects in VFS

- Superblock Object: representing a mounted filesystem.
- Inode Object: representing a file (device/FIFO/socket) in filesystem.
- Directory Entry Object: representing a directory entry (dentry).
- File Object: representing an opened file in process.

Operations/Structs of VFS Object

- *super_operations*: how the VFS can manipulate the superblock of your filesystem
- *inode_operations*: how the VFS can manipulate an inode in your filesystem
- *dentry_operations*: how a filesystem can overload the standard dentry operations
- *file_operations*: how the VFS can manipulate an open file

Portable Operating System Interface



Topics

[Batch Services](#)

[Development](#)

[Headers](#)

[Math Interfaces](#)

[Networking](#)

[Realtime](#)

[Threads](#)

The **POSIX** is a family of standards specified by the IEEE Computer Society for maintaining compatibility between operation systems.^[1]

POSIX defines both the system- and user-level application programming interfaces (API), along with command line shells and utility interfaces, for software compatibility (portability) with variants of Unix and other operating systems.^{[2][3]}

POSIX is also a trademark of the IEEE.^[2] POSIX is intended to be used by both application and system developers.

[Version: POSIX.1-2017](#)

INDEX

[[Alphabetic](#) | [Topic](#) | [Word Search](#)]

Select a Volume:

[[Base Definitions](#) | [System Interfaces](#) | [Shell & Utilities](#) | [Rationale](#)]

[[Frontmatter](#)]

[[Main Index](#)]

System Interfaces

1. [Introduction](#)
2. [General Information](#)
3. [System Interfaces](#)

[<<< Previous](#)

[Home](#)

The Open Group Base Specifications Issue 7, 2018 edition
IEEE Std 1003.1-2017 (Revision of IEEE Std 1003.1-2008)
Copyright © 2001-2018 IEEE and The Open Group

1. Introduction

The System Interfaces volume of POSIX.1-2017 describes the interfaces offered to application programs by POSIX-conformant systems.

1.1 Relationship to Other Formal Standards

Great care has been taken to ensure that this volume of POSIX.1-2017 is fully aligned with the following standards:

ISO C (1999)

ISO/IEC 9899:1999, Programming Languages - C, including ISO/IEC 9899:1999/Cor.1:2001(E), ISO/IEC 9899:1999/Cor.2:2004(E)

Parts of the ISO/IEC 9899:1999 standard (hereinafter referred to as the ISO C standard) are referenced to describe requirements also in headers included within this volume of POSIX.1-2017 have a version in the ISO C standard; in this case CX markings are added as appropriate (see [Codes](#)). Any conflict between this volume of POSIX.1-2017 and the ISO C standard is unintentional.

This volume of POSIX.1-2017 also allows, but does not require, mathematics functions to support IEEE Std 754-1985 and IEEE Std 854-1

1.2 Format of Entries

The entries in [System Interfaces](#) are based on a common format as follows. The only sections relating to conformance are the SYNOPSIS,

NAME

This section gives the name or names of the entry and briefly states its purpose.

SYNOPSIS

This section summarizes the use of the entry being described. If it is necessary to include a header to use this function, the names

```
#include <stdio.h>
```

INDEX

[[Alphabetic](#) | [Topic](#) | [Word Search](#)]

Select a Volume:

[[Base Definitions](#) | [System Interfaces](#) |
[Shell & Utilities](#) | [Rationale](#)]

[[Frontmatter](#)]

[[Main Index](#)]

[<<< Previous](#)

[Home](#)



[Next >>>](#)

The Open Group Base Specifications Issue 7, 2018 edition
IEEE Std 1003.1-2017 (Revision of IEEE Std 1003.1-2008)
Copyright © 2001-2018 IEEE and The Open Group

NAME

open, openat - open file

SYNOPSIS

```
[OH]  #include <sys/stat.h>  
#include <fcntl.h>
```

```
int open(const char *path, int oflag, ...);  
int openat(int fd, const char *path, int oflag, ...);
```

DESCRIPTION

The *open()* function shall establish the connection between a file and a file descriptor. It shall create an open file description that refers to a file and a file descriptor that refers to that open file description. The file descriptor is used by other I/O functions to refer to that file. The *path* argument points to a pathname naming the file.

The *open()* function shall return a file descriptor for the named file, allocated as described in [File Descriptor Allocation](#). The open file description is new, and therefore the file descriptor shall not share it with any other process in the system. The FD_CLOEXEC file descriptor flag associated with the new file descriptor shall be cleared unless the O_CLOEXEC flag is set in *oflag*.

The file offset used to mark the current position within the file shall be set to the beginning of the file.

The file status flags and file access modes of the open file description shall be set according to the value of *oflag*.

Values for *oflag* are constructed by a bitwise-inclusive OR of flags from the following list, defined in [<fcntl.h>](#). Applications shall specify exactly one of the first five values (file access modes) below in the value of *oflag*:

System Interfaces

1. [Introduction](#)
2. [General Information](#)
3. [System Interfaces](#)

Demo

Create One File as Your Virtual Disk/Dev

```
zegang@zegang-VirtualBox:~$ mkdir 1300-simple-fs
zegang@zegang-VirtualBox:~$ cd 1300-simple-fs/
zegang@zegang-VirtualBox:~/1300-simple-fs$ touch mydisk
zegang@zegang-VirtualBox:~/1300-simple-fs$ truncate -s 128M mydisk
zegang@zegang-VirtualBox:~/1300-simple-fs$ ll -h mydisk
-rw-rw-r-- 1 zegang zegang 128M 2月 24 17:33 mydisk
```

Format the Disk for EXT4 file system

```
zegang@zegang-VirtualBox:~/1300-simple-fs$ mkfs.ext4 mydisk
mke2fs 1.45.5 (07-Jan-2020)
Discarding device blocks: done
Creating filesystem with 32768 4k blocks and 32768 inodes

Allocating group tables: done
Writing inode tables: done
Creating journal (4096 blocks): done
Writing superblocks and filesystem accounting information: done
```

Demo

Make an empty directory as future mount point

```
zegang@zegang-VirtualBox:~/1300-simple-fs$ mkdir mydisk_mp
```

Mount Your Virtual Device

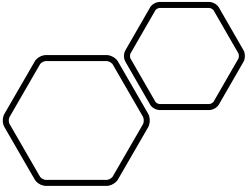
```
zegang@zegang-VirtualBox:~/1300-simple-fs$ ll -dh mydisk_mp
drwxrwxr-x 2 zegang zegang 4.0K 2月 24 17:38 mydisk_mp/
zegang@zegang-VirtualBox:~/1300-simple-fs$ sudo mount mydisk mydisk_mp
[sudo] password for zegang:
```

Info of Your mounted File System

```
zegang@zegang-VirtualBox:~/1300-simple-fs$ findmnt mydisk_mp
TARGET                                SOURCE          FSTYPE OPTIONS
/home/zegang/1300-simple-fs/mydisk mp /dev/loop14 ext4  rw,relatime
```

Default owner of root directory

```
zegang@zegang-VirtualBox:~/1300-simple-fs/mydisk_mp$ ll -hd .
drwxr-xr-x 4 root root 4.0K 2月 24 17:55 ./
```



Demo

- Create directories and files on Your Virtual Disk

```
zegang@zegang-VirtualBox:~/1300-simple-fs$ cd mydisk_mp/
zegang@zegang-VirtualBox:~/1300-simple-fs/mydisk_mp$ sudo mkdir -p home/zegang
zegang@zegang-VirtualBox:~/1300-simple-fs/mydisk_mp$ ll home/
total 12
drwxr-xr-x 3 root root 4096 2月 24 17:55 ./
drwxr-xr-x 4 root root 4096 2月 24 17:55 ../
drwxr-xr-x 2 root root 4096 2月 24 17:55 zegang/
zegang@zegang-VirtualBox:~/1300-simple-fs/mydisk_mp$ chown -R zegang home/zegang
chown: changing ownership of 'home/zegang': Operation not permitted
zegang@zegang-VirtualBox:~/1300-simple-fs/mydisk_mp$ sudo chown -R zegang home/zegang
zegang@zegang-VirtualBox:~/1300-simple-fs/mydisk_mp$ ll home/
total 12
drwxr-xr-x 3 root root 4096 2月 24 17:55 ./
drwxr-xr-x 4 root root 4096 2月 24 17:55 ../
drwxr-xr-x 2 zegang root 4096 2月 24 17:55 zegang/
zegang@zegang-VirtualBox:~/1300-simple-fs/mydisk_mp$ echo "1" > home/zegang/1.txt
zegang@zegang-VirtualBox:~/1300-simple-fs/mydisk_mp$ cat home/zegang/1.txt
1
```

References

- Wiki Filesystem, https://en.wikipedia.org/wiki/File_system
- Inode, <https://en.wikipedia.org/wiki/Inode>
- INTRODUCTION TO THE LINUX VIRTUAL FILESYSTEM,
[HTTPS://WWW.STARLAB.IO/BLOG/INTRODUCTION-TO-THE-LINUX-VIRTUAL-FILESYSTEM-VFS-PART-I-A-HIGH-LEVEL-TOUR](https://www.starlab.io/blog/introduction-to-the-linux-virtual-filesystem-vfs-part-i-a-high-level-tour)